



C# Avançado

WPF Parte 2: Manipulando Dados

Softblue
cursos online

Tópicos Abordados

- Binding
 - Binding modes
 - Tipos de bindings
 - Entre controles WPF
 - Com elementos estáticos
 - Com objetos
 - Com coleções de objetos
- Conversão de dados
 - Formatação de strings
 - Conversão customizada
- Validação de dados
 - Lançamento de exceção
 - Interface *INotifyDataErrorInfo*
 - Validadores customizados

Tópicos Abordados

- Padrão MVVM (Model-View-ViewModel)
 - Comandos em WPF
 - Estrutura

Binding



- O processo de binding consiste em atrelar uma fonte de dado a um destino
- Por exemplo, quando dados vindos do banco de dados precisam ser mostrados na tela

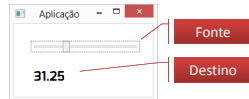


- A fonte pode ser qualquer uma
- O destino normalmente são controles WPF

Binding Entre Controles WPF



- Uma das formas de binding consiste em atrelar uma propriedade de um controle a uma propriedade de outro controle
 - Quando a fonte muda, o destino é atualizado automaticamente



```
Fonte
<Slider x:Name="slider" Minimum="0" Maximum="100"/>

Destino
<Label Content="{Binding ElementName=slider, Path=Value}" />
```

Lendo Bindings Via Programação



- A origem do binding pode ser obtida via programação

```
Fonte
<Slider x:Name="slider" Minimum="0" Maximum="100"/>

Destino
<Label x:Name="label" Content="{Binding ElementName=slider, Path=Value}" />

BindingExpression exp =
    BindingOperations.GetBindingExpression(label, Label.ContentProperty);
Slider slider = (Slider)exp.ResolvedSource;
```

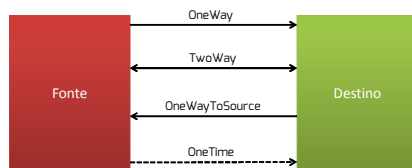
Retorna a fonte do binding

Binding Modes



- Um binding pode ser configurado para usar um dos modos abaixo do enum *BindingMode*
 - *OneWay*
 - *TwoWay*
 - *OneWayToSource*
 - *OneTime*
 - *Default*
- Quando um *BindingMode* não é especificado, *Default* é assumido

Binding Modes




- O binding mode *Default* depende da propriedade de destino
 - Propriedades editáveis costumam ser ***TwoWay*** por padrão (Ex: *TextBox.Text*, *CheckBox.IsChecked*, etc.)
 - Propriedades não editáveis costumam ser *OneWay* por padrão (Ex: *Label.Text*, *Button.Content*, etc.)

Atualização dos Dados na Fonte



- Bindings que operam nos modos *TwoWay* e *OneWayToSource* precisam atualizar dados da fonte
- Em que momento isso deve ser feito?
- O binding pode ser configurado através do enum *UpdateSourceTrigger*
 - *PropertyChanged*
 - *LostFocus*
 - *Explicit*
 - *Default*

Atualização dos Dados na Fonte



- A atualização no modo *Explicit* deve ser feita via programação

```


BindingExpression exp =
    BindingOperations.GetBindingExpression(target, TextBox.TextProperty);
exp.UpdateSource();
    
```

Destino do binding

Atualiza a fonte

- O modo *Default* depende de cada controle
 - Normalmente, o padrão é *PropertyChanged*
 - No *TextBox*, o padrão é *LostFocus*

Binding com Elementos Estáticos



- Elementos estáticos também podem ser utilizados no processo de binding

```

<Window x:Class="App.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:n="clr-namespace:App">


    <Window.Resources>
        <ObjectDataProvider x:Key="consts" ObjectType="{x:Type n:Constants}" />
    </Window.Resources>
    ...
    <TextBlock Text="{Binding
        Source={StaticResource ResourceKey=consts}, Path=MSG}" />
    
```

Referencia o namespace da classe (App)

Chama a classe Constants de consts


Acessa Constants.MSG

Binding com Objetos



- É o tipo de binding mais utilizado em aplicações WPF
- Permite fazer o binding entre dados da aplicação (vindos do banco de dados, por exemplo) e controles em uma janela WPF

Exemplo de Binding com Objetos



Arquivo XAML

```

<Grid Name="gridProduto">
  <TextBox Text="{Binding Path=Nome}" />
  <TextBox Text="{Binding Path=Valor}" />
  <TextBox Text="{Binding Path=Descricao}" />
</Grid>

```

Binding para as properties do objeto

Code Behind File


```

Produto p = Dao.ObterProduto(10);
gridProduto.DataContext = p;

```


Atrala o produto ao Grid

A Interface *INotifyPropertyChanged*



- Normalmente utilizada em situações onde ocorre o binding com objetos
- Avisa o WPF que uma ou mais propriedades do objeto da fonte foram alteradas
 - Assim o WPF consegue atualizar os dados na tela

A Interface *INotifyPropertyChanged*



Implementa a interface *INotifyPropertyChanged*

```

class Produto : INotifyPropertyChanged
{
  public event PropertyChangedEventHandler PropertyChanged;

  string nome;

  public string Nome
  {
    get { return nome; }
    set
    {
      this.nome = value;
      if (PropertyChanged != null)
      {
        PropertyChanged(this, new PropertyChangedEventArgs("Nome"));
      }
    }
  }
}

```

Evento disparado quando uma propriedade é alterada

Disparo do evento

Binding com Coleções de Objetos



- Alguns controles do WPF permitem o binding a uma coleção de objetos
 - Controles que herdam de *ItemsControl*
 - *ListBox*, *ComboBox*, *DataGrid*, etc.
- A fonte pode ser qualquer coleção de dados
 - Interface *IEnumerable*
- A propriedade *ItemsSource* define o binding

```
List<Produto> produtos = Dao.GetProducts();  
prodList.ItemsSource = produtos;
```

Configurando a Exibição



- Cada elemento da coleção precisa ser exibido de uma forma pelo controle do WPF
- O que vai ser mostrado pode ser configurado de 3 formas
 - Através da sobrescrita do método *ToString()* na classe dos objetos
 - Através da propriedade *DisplayMemberPath* do controle WPF
 - Através do uso de templates

A Classe *ObservableCollection<T>*



- O binding com uma coleção do tipo *IEnumerable* não avisa o WPF a respeito de mudanças na coleção
 - Inserção ou exclusão de elementos
- Para estas situações, a classe *ObservableCollection<T>* pode ser usada
 - Se ocorrer uma mudança na coleção, o controle WPF atrelado à coleção será avisado e a sincronização será feita

Conversão de Dados

- É comum que dados precisem ser convertidos ao serem exibidos pelo controle WPF ou na atualização da fonte

Produto
Valor = 120.2

→

R\$ 120,20

- A conversão pode ser feita de duas formas
 - Formatação de strings
 - Conversores customizados

Conversão com Formatação de String

- A propriedade *StringFormat* do binding é utilizada

```

<TextBox Text="{Binding Path=Valor, StringFormat={}{0:C}}" />
  
```

- Outro exemplo

```

<TextBlock Text="{Binding Path=Valor, StringFormat=O valor é {0:C}}" />
  
```

Abordagem ideal para conversão numérica e de datas

Conversão Customizada

- A conversão pode ser feita entre quaisquer tipos de objetos
- É preciso criar um classe que implementa *IValueConverter*

```

[ValueConversion(typeof(Cpf), typeof(string))]
class CpfConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        Cpf cpf = (Cpf)value;
        return cpf.Numero;
    }

    public object ConvertBack(object value, Type targetType, object
        parameter, CultureInfo culture)
    {
        string str = (string)value;
        return new Cpf() { Numero = str };
    }
}
  
```

Opcional

Converte do formato nativo para o formato de exibição

Converte do formato de exibição para o formato nativo

Conversão Customizada



- A conversão é aplicada no arquivo XAML

```
<Window x:Class="App.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:App">
    ...
    <TextBox>
        <TextBox.Text>
            <Binding Path="Cpf">
                <Binding.Converter>
                    <local:CpfConverter />
                </Binding.Converter>
            </Binding>
        </TextBox.Text>
    </TextBox>
</Window>
```

Referencia o namespace da aplicação

Referencia a classe de conversão customizada

Validação de Dados




- Validar dados é uma ação importante em aplicações que permitem a interação com usuários
- Existem diversas técnicas de validação de dados em C#
 - Cada técnica pode ser usada em um tipo de cenário

Validação com Lançamento de Exceção



- O objeto que contém os dados pode lançar exceções quando uma regra de validação for violada
- O WPF altera o visual do controle para mostrar que houve um problema
- Técnica utilizada quando o próprio objeto já incorpora as regras de validação nas suas properties

Validação com Lançamento de Exceção



```

class Produto
{
    string nome;

    public string Nome
    {
        get { return nome; }
        set
        {
            if (string.IsNullOrEmpty(value))
            {
                throw new ArgumentOutOfRangeException("Valor inválido");
            }
            nome = value;
        }
    }
}

```

Regra de validação


```

...
<TextBox Margin="5" Grid.Column="1">
    <TextBox.Text>
        <Binding Path="Nome">
            <Binding.ValidationRules>
                <ExceptionValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
...

```


Habilita o reconhecimento de exceções

A Interface *INotifyDataErrorInfo*



- Pode ser implementada pela classe dos dados
- As regras de validação continuam dentro do objeto
- Não é necessário fazer o lançamento de exceções

A Interface *INotifyDataErrorInfo*



```

class Produto : INotifyDataErrorInfo
{
    public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;
    public IEnumerable GetErrors(string propertyName) { }
    public bool HasErrors { }
}

```

Evento disparado quando a coleção de erros é alterada

Indicativo da existência de erros

Coleção de erros de uma propriedade

Os erros costumam ser armazenados em um *Dictionary<string, List<T>>*

Validadores Customizados



- A validação também pode ser feita por validadores customizados
- O código de validação fica separado dos dados que serão validados
 - Permite a reutilização da lógica de validação
- A classe de validação deve estender a classe *ValidationRule*

Exemplo de Validador



```
class MinValueValidator : ValidationRule
{
    public int Min { get; set; }

    public MinValueValidator()
    {
        Min = 0;
    }

    public override ValidationResult Validate(object value, CultureInfo cultureInfo)
    {
        try
        {
            number = int.Parse((string)value);
            if (number < Min)
            {
                return new ValidationResult(false, "Número menor que " + Min);
            }
        }
        catch
        {
            return new ValidationResult(false, "O número é inválido");
        }
        return new ValidationResult(true, null);
    }
}
```

Property para
customizar o validador

Exemplo de Validador



- O arquivo XAML deve referenciar o validador customizado

```
...
<TextBox>
    <TextBox.Text>
        <Binding Path="Valor">
            <Binding.ValidationRules>
                <local:MinValueValidator Min="100" />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
...
```

Se mais de um validador for especificado, a ordem de execução é de cima para baixo

O processo de validação ocorre antes do processo de conversão

Evento de Erro de Validação



- Um evento pode ser disparado quando ocorre um erro de validação
- É preciso que o binding suporte o disparo deste evento

```
<Grid Name="gridProduto" Validation.Error="OnError">  
  <Binding Path="..." NotifyOnValidationError="True">  
</Grid>
```

Método
chamado

Ativa o disparo
do evento

```
private void OnError(object sender, ValidationErrorEventArgs args)  
{  
  //...  
}
```

Padrão MVVM



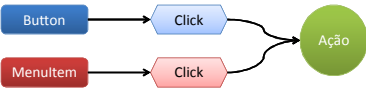

- O padrão MVVM costuma ser bastante utilizado na construção de aplicações WPF
 - MVVM: Model-View-ViewModel
- Variante do padrão MVC (Model-View-Controller)
- O MVVM se apoia no uso de recursos do WPF
 - Binding
 - Command

Usando Comandos



- O uso de comandos é uma alternativa ao uso de alguns tipos de eventos
- Um evento é atrelado a um controle específico, enquanto um comando pode ser atrelado a vários controles
 - Isto permite uniformizar a execução de ações
- Um comando pode ter sua execução habilitada ou desabilitada
 - Os controles ligados ao comando refletem este estado

Eventos x Comandos

- Usando eventos
 
- Usando comandos
 

Interface *ICommand*

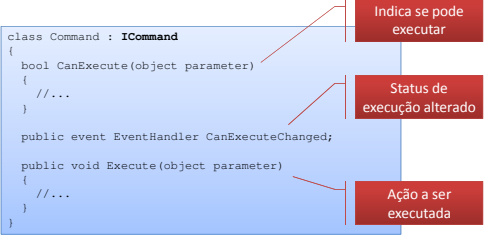
- Um comando deve implementar a interface *ICommand*

```

class Command : ICommand
{
    bool CanExecute(object parameter)
    {
        //...
    }

    public event EventHandler CanExecuteChanged;

    public void Execute(object parameter)
    {
        //...
    }
}
  
```



Comando no XAML

- Controles que suportam o uso de comandos podem ter um comando atrelado através do atributo *Command*

```

<Button Content="Processar" Command="..." />
  
```

